Attorney Docket No.: 021202-003710US
Client Reference No.: QST-088-1 CV

PATENT APPLICATION

ADAPTABLE DATAPATH FOR A DIGITAL PROCESSING SYSTEM

Inventor: Amit Ramchandran, a citizen of India, residing at
6082 Monterey Road, #204
San Jose, CA 95119

Assignee: QuickSilver Technology, Inc.
6640 Via Del Oro, Suite 120
San Jose, CA 95119

Entity: Small business concern

CARPENTER AND KULAS, LLP
1900 Embarcadero Road
Suite 109
Palo Alto, CA 94303
Tel: 650-842-0300
Fax: 650-842-0304

ADAPTABLE DATAPATH FOR A DIGITAL PROCESSING SYSTEM

CLAIM OF PRIORITY

This application claims priority from U.S. Provisional Patent Application Serial No. 60/422,063, filed Oct. 28, 2002; entitled "RECONFIGURATION NODE RXN" which is hereby incorporated by reference as if set forth in full in this application.

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to the following co-pending U.S. Patent Applications that are each incorporated by reference as if set forth in full in this application:

"INPUT PIPELINE REGISTERS FOR A NODE IN AN ADAPTIVE COMPUTING ENGINE," Serial No. _____ [TBD], filed _____[TBD] (Our Ref. No. 021202-003720US);

"CACHE FOR INSTRUCTION SET ARCHITECTURE USING INDEXES TO ACHIEVE COMPRESSION," Serial No. _____ [TBD], filed _____[TBD] (Our Ref. No. 021202-003730US);

"METHOD FOR ORDERING OPERATIONS FOR SCHEDULING BY A MODULO SCHEDULER FOR PROCESSORS WITH A LARGE NUMBER OF FUNCTION UNITS AND RECONFIGURABLE DATA PATHS," Serial No. 10/146,857, filed on May 15, 2002 (Our Ref. No. 021202-002700US);

"UNIFORM INTERFACE FOR A FUNCTIONAL NODE IN AN ADAPTIVE COMPUTING ENGINE," Serial No. _____ [TBD], filed on May 21, 2003 (Our Ref. No. 021202-003400US);

"HARDWARE TASK MANAGER FOR ADAPTIVE COMPUTING," Serial No. _____ [TBD], filed on May 21, 2003 (Our Ref. No. 021202-003500US);

"ADAPTIVE INTEGRATED CIRCUITRY WITH HETEROGENEOUS AND RECONFIGURABLE MATRICES OF DIVERSE AND ADAPTIVE COMPUTATIONAL UNITS HAVING FIXED, APPLICATION SPECIFIC COMPUTATIONAL ELEMENTS," Serial No. 09/815,122, filed on March 22, 2001;

BACKGROUND OF THE INVENTION

This invention is related in general to digital processing architectures and more specifically to the use of a adaptable data path using register files to efficiently implement digital signal processing operations.

Digital signal processing (DSP) calculations require many iterations of fast multiply-accumulate and other operations. Typically, the actual operations are accomplished by "functional units" such as multipliers, adders, accumulators, shifters, etc. The functional units obtain values, or operands, from a fast main memory such as random access memory (RAM). The DSP system can be included within a chip that resides in a device such as a consumer electronic device, computer, etc.

The design of a DSP chip can be targeted for specific DSP applications. For example, in a cellular telephone, a DSP chip may be optimized for time-division multiple access (TDMA) processing. A voice-over-internet protocol (VOIP) application may require vocoding operations, and so on. It is desirable for a chip manufacturer to provide a single chip design that can be adapted to different DSP applications. Such a chip is often described as an adaptable, or configurable, design.

One aspect of an adaptable design for a DSP chip includes allowing flexible and configurable routing between the different functional units, memory and other components such as registers, input/output and other resources on the chip. A traditional approach to providing flexible routing uses a data bus. Such an approach is shown in Fig. 1.

In Fig. 1, memory bus 10 interfaces with a memory (not shown) to provide values from the memory to processing components such as functional unit blocks 30, 32 and 34. Values from memory bus 10 are selected and routed through memory bus interface 20 to data path bus 36. The functional unit blocks are able to obtain values from data path bus 36 by using traditional bus arbitration logic (e.g., address lines, bus busy, etc.). Within a block, such as functional unit block 30 of Fig. 1, there may be many different components, such as a bank of multipliers, to which the data from data path bus 36 can be transferred. In this manner, any arbitrary value from memory can be provided to any functional unit block, and to components within blocks of functional units.

Values can also be provided between functional unit blocks by using the data path bus. Another resource is register file 60 provided on data path bus 36 by register file interface 50.

Register file 60 includes a bank of fast registers, or fast RAM. Register file interface 50 allows values from data path bus 36 to be exchanged with the register file. Typically, any register or memory location within register file 60 can be placed on data path bus 36 within the same amount of time (e.g., a single cycle). One way to do this is to provide an address to a location in the register file, either on the data path bus, itself, or by using a separate set of address lines. This approach is very flexible in that any value in a component of a functional unit block can be transferred to any location within the register file and vice versa.

However, a drawback with the approach of Fig. 1, is that such a design is rather expensive to create, slow and does not scale well. A bus approach requires considerable overhead in control circuitry and arbitration logic. This takes up real estate on the silicon chip and increases power consumption. The use of a large, randomly addressable register file also is quite costly and requires inclusion of tens of thousands of additional transistors. The use of such complicated logic often requires bus cycle times to be slower to accommodate all of the switching activity. Finally, such an approach does not scale well since, e.g., adding more and more functional unit blocks will require additional addressing capability that may mean more lines and logic. Additional register file space may also be required. The data path bus would also need to be routed to connect to the added components. Each functional unit block also requires the bus control and arbitration circuitry.

Thus, it is desirable to provide an interconnection scheme for digital processor applications that improves over one or more of the above, or other, shortcomings in the prior art.

SUMMARY OF THE INVENTION

The present invention uses dedicated groups of configurable data path lines to transfer data values from a main memory to functional units. Each group of data path lines includes a register file dedicated for storage for each group of lines. Functional units can obtain values from, and store values to, main memory and can transfer values among the registers and among other functional units by using the dedicated groups of data path lines and a data address generator (DAG).

DAG circuitry interfaces each group of datapath lines to a main memory bus. Each DAG is controllable to select a value of varying bit width from the memory bus, or to select a value from another group of data path lines. In a preferred embodiment, eight groups of 16 data

path lines are used. Each group includes a register file of eight 16-bit words on each group of 16 data path lines. Registers can hold a value onto their associated group of data path lines so that the value is available at a later time on the lines without the need to do a later data fetch.

In one embodiment the invention provides a data path circuit in a digital processing device, wherein the data path circuit is coupled to a memory bus for obtaining values from a memory, the data path circuit comprising a first plurality of data lines; a first data address generator for coupling the first plurality of data lines to the memory bus so that a value from the memory transferred by the memory bus can be placed onto the first plurality of data lines; one or more functional units for performing a digital operation coupled to the plurality of data lines; and a register coupled to the first plurality of data lines, wherein the register selectively stores a value from the first plurality of data lines so that the value is selectively available on the first plurality of data lines.

Another aspect of the invention provides both general and direct data paths between array multipliers and accumulators. Banks of accumulators are coupled to the groups of configurable data path lines and are also provided with direct lines to the multipliers. An embodiment of the invention provides a digital processing system comprising a multiplier; an accumulator; a configurable data path coupled to the multiplier and the accumulator; and a direct data path coupled between the multiplier and the accumulator.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a prior art approach using a data path bus;

Fig. 2 illustrates the configurable data path arrangement of the present invention;

DETAILED DESCRIPTION OF THE INVENTION

A preferred embodiment of the invention is incorporated into a node referred to as a Adaptable Node (RXN) in a adaptive computing engine (ACE) manufactured by Quicksilver, Inc., of San Jose, California. Details of the ACE engine and RXN node can be found in the priority and related patent applications reference above. Aspects of the invention described herein are adaptable for use with any generalized digital processing system, such as a system adapted for digital signal processing or other types of processing.

Fig. 2 illustrates the configurable data path arrangement of the present invention.

In Fig. 2, digital processing system 100 is designed for fast DSP-type processing such as in discrete cosine transformation (DCT), fast fourier transformation (FFT), etc. Digital processing system 100 includes four 32-bit data path address generators (DAG) to interface between four groups of configurable data path lines 200 and a main memory bus 110. Main memory bus 110 is an arbitrated high-speed bus as is known in the art. Other types of main memory accessing can be used.

Each group of 32 lines includes two subgroups of 16 lines each. Each subgroup connected to a register files of eight 16-bit words. For example, DAG 120 is connected to register files 180 and 182. DAG 122 is connected to register files 184 and 186. Similarly, DAGs 124 and 126 are connected to register files 188, 190 and 192, 194, respectively. Naturally, other embodiments can use any number of DAGs, groups, subgroups register files. Although specific bit widths, numbers of lines, components, etc., and specific connectivity are described, many variations are possible and are within the scope of the invention. Although the DAGs play a major role in the preferred embodiment, other embodiments can use other types of interfacing to the main memory bus. Although the DAGs provide a high degree of configurable routing options (as discussed below), other embodiments can vary in the degree of configurability, and in the specific configuration options and control methods. In some cases simple registers, register files, multiplexers or other components might be used in place of the DAGs of the present invention.

The use of register files on each of the discrete subgroup lines simplifies the interconnection architecture from that of the more generalized bus and multiport register file shown in Fig. 1 of the prior art. This approach can also provide benefits in reduced transistor count, power consumption, improved scalability, efficient data access and other advantages. Although configuring the data path of the present invention may be more complex than with generalized approaches, in practice a compiler is able to automatically handle the configuration transparently to a human programmer. This allows creation of faster-executing code for a variety of DSP applications by using the same hardware architecture without any placing any undue burden on the programmer. If desired, a programmer can customize the data path configuration in order to further optimize processing execution.

Groups of data path lines 200 are used to transfer data from memory bus 110 to functional units within blocks 130 and 132, and also to transfer data among the functional units, themselves. The functional unit blocks are essentially the same so only block 130 is discussed in detail. Functional units include programmable array multipliers (PAMs) 140, accumulators (and shift registers)150, data cache 160 and arithmetic/logic units (ALUs) 170 and 172. Naturally, the functional units used in any specific embodiment can vary in number and type from that shown in Fig. 2.

Functional units are connected to the data path line groups via multiplexers and demultiplexers such as 210 and 220, respectively. Inputs and outputs (I/Os) from the functional units can, optionally, use multiplexing to more than one subgroup of data path lines; or an I/O can be connected directly to one subgroup. A preferred embodiment uses pipeline registers between I/O ports and data path lines, as shown by boxes labeled "p" in Fig. 2. Pipeline registers allow holding data at I/O ports, onto data lines, or for other purposes. The pipeline registers also allow obtaining a zero, 1, or other desired binary values and provide other advantages. Pipeline registers are described in more detail in the co-pending patent application "INPUT PIPELINE REGISTERS FOR A NODE IN AN ADAPTIVE COMPUTING ENGINE" referenced above.

Table I, below, shows DAG operations. The configuration of the data path from cycle to cycle is set by a control word, or words obtained from the main memory bus in accordance with controller modules such as a hardware task manager, scheduler and other processes and components not shown in Fig. 2 but discussed in related patent applications. Part of the configuration information includes fields for DAG operations. A DAG operation can change from cycle to cycle and includes reading data of various widths from memory or from another DAG. DAG operations other than those shown in Table I can be used. Each DAG has one 5-bit 'dag-op' field and one 4-bit 'address' field. There is a single 'pred' field that defines non-sequencing operations.

Dag- p	Mnemonic	Descripti n	Cycles
0x00	read8	Read 8-bits from memory	1
0x01	read8x	Read 8-bits from memory and sign extend to 32-bits	1
0x02	read16	Read 16-bits from memory	1
0x03	read16x	Read 16-bits from memory and sign extend to 32-bits	1
0x04	read24	Read 24-bits from memory	1
0x05	read24x	Read 24-bits from memory and sign extend to 32-bits	1
0x06	read32	Read 32-bits from memory	1
0x07	write8	Write 8-bits to memory	1
0x08	write16	Write 16-bits to memory	1
0x09	write24	Write 24-bits to memory	1
0x0A	write32	Write 32-bits to memory	1
0x0B	writeMindp	Write 32-bits (only mode supported) to MIN write queue from the data path buses	1
0x0C	writeMinM	Write 32-bits (only mode supported) to MIN write queue from a 32-bit memory read.	1 (pipelined)
0x0D	readdag16	Read a 16 bit value from one DAG register	0
0x0E	readdag32	Read a 32 bit value from two DAG registers	0
0x0F	load32dp	Load two 16-bit DAG registers or 32-bit write buffer using 32-bit data in dp2n:dp2n+1 connecting to DAGn	1
0x10	load16dpn	Load a DAG register from an even data path bus	1
0x11	load16dpn+1	Load a DAG register from an odd data path bus	1
0x12	modify	Modify address but do not do a memory access.	1
0x13	Dagnoop	Do nothing. All DAG operations execute every clock cycle until this operation is chosen	1
0x14	Dagcont	Continue the previous operation	1
0x15	writePA	Writes 32-bits of data from memory into 'tfrl' or 'tbrl'	1
0x16	writeMinbuf	Write 32-bits to MIN write queue from buffer	1

TABLE I

For dag-op: 0x00 to 0x0A, 0x0C and 0x12 the DAG operation format of Table II applies. The address field is divided into action and context as shown.

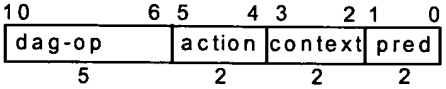


TABLE II

Action

The 'action' field describes the address modification/generation process using a set of registers (base, limit, index and delta) pointed to by the 'context' field.

Table 1: DAG address calculation

action	Operation	Description
00	Supply an address and post modify	Address = Base + Index Index = Index + delta (delta is a signed value) If Index >= limit, Index = Index – limit If Index < 0, Index = limit + Index
01	Supply a pre-modified address	Index = Index + delta (delta is a signed value) If Index >= limit, Index = Index – limit If Index < 0, Index = limit + Index Address = Base + Index
11	Supply a bit-reversed address	Address = Base + B-Index B-Index = reverse carry add (Index + delta) (delta is a signed value) If Index >= limit, Index = Index – limit If Index < 0, Index = limit + Index

Context

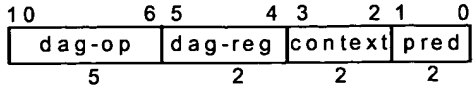
The 'context' field is used to point at a specific DAG setting (base, limit, index and delta) on which an 'action' is performed or a DAG register is accessed (II)

Table 2: context encoding

context	Operation
00	Use setting - basen.0, limitn.0, indexn.0, deltan.0 for DAGn
01	Use setting - basen.1, limitn.1, indexn.1, deltan.1 for DAGn
10	Use setting - basen.2, limitn.2, indexn.2, deltan.2 for DAGn
11	Use setting - basen.3, limitn.3, indexn.3, deltan.3 for DAGn

For convenience, an ACTION function is defined according to the action table – ACTION (action, context) where 'action' and 'context' refer to the DAG operation fields. This function is used in the individual DAG operation descriptions.

(II) For dag-op: 0x0D to 0x11 the following DAG operation format applies:



The 'dag-reg' field is used to identify a specific 16-bit register (base or limit or index or delta) within a DAG 'context' as specified by the dag-reg table (below)

Table 3: dag-reg encoding for dag-op 0x0D, 0x10 and 0x11

dag-reg	Register
00	base
01	limit
10	index
11	delta

For operations 0x0E and 0x0F, the dag-reg field is used to address 2 DAG registers – base and limit or index and delta or a write buffer location. In this case, the 'dag-reg' table is as follows:

Table 4: dag-reg encoding for dag-op 0x0E

dag-reg	Register
0X	Base and limit
1X	Index and delta

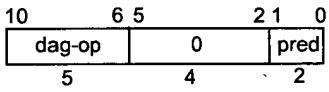
X – don't care

Table 5: dag-reg encoding for dag-op 0x0F

dag-reg	Register
00	Base and limit
10	Index and delta
11	Location 'n' of write buffer for DAGn

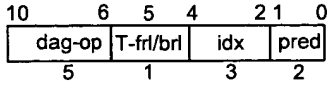
01 - undefined

(III) For dag-op: 0x0B, 0x13, 0x14, and 0x16 the following DAG operation format applies:



The address field in this case is unused, which is represented as "0" in the RXN.

(IV) For dag-op: 0x15 the following DAG operation format applies:



The 'T-frl/brl' field is used to choose between the translation frl and the translation brl

T- frl/brl	Operation
0	The 'idx' field points to T-frl
1	The 'idx' field points to T-brl

The T-frl and T-brl each have 5 32-bit locations. The 'idx' field is used to address these five locations

add	Operation
000	Location 0
001	Location 1
010	Location 2
011	Location 3
100	Location 4
101	Location 5

Pred

The universal 'pred' field along with the 's' bit determines whether a DAG operation is executed or not executed. When a DAG operation is 'not executed' due to its predication, the last executed DAG operation executes again.

Table 6: Pred field encoding

Pred	Description
00	Never execute
01	Always execute (execute specified operations)
10	Execute if condition is true ("s" bit is set) (execute specified operation) If condition is false ("s" bit is not set) (do not execute the DAG operation)
11	Execute if condition is false ("s" bit is not set) (execute specified operation) If condition is true ("s" bit is set) (do not execute the DAG operation)

Note1: All DAG operations execute every clock cycle until "dagnoop" operation is chosen.

Although the invention has been discussed with respect to specific embodiments thereof, these embodiments are merely illustrative, and not restrictive, of the invention. For

example, although the node has been described as part of an adaptive computing machine, or environment, aspects of the filter node design, processing and functions can be used with other types of systems. In general, the number of lines and specific interconnections can vary in different embodiments. Specific components, e.g., the data address generator, can be implemented in different ways in different designs. Components may be omitted, substituted or implemented with one or more of the same or different components. For example, a data address generator can by substituted with a general register, or it can be a different component responsive to a control word. Many variations are possible.

Thus, the scope of the invention is to be determined solely by the dependent claims.